Front Page

1. INTRODUCTION

1.1 PROJECT OVERVIEW

An inventory management system (IMS) is a computerized program that helps businesses track and manage their inventories. An IMS enables businesses to reduce the costs associated with inventory, and improve the accuracy and timeliness of supply. Businesses use an IMS for different types of goods including raw materials, supplies, and finished products. Most IMS programs include features for managing purchase orders, sales orders, and shipping and receiving. Many systems also include reporting capabilities such as sales and operations planning reports (S&OP) and monthly business reviews (MBR). Some systems are also designed for specific industries, such as healthcare and manufacturing. Retail IMS systems are geared toward small businesses with limited capital and resources and can automate the ordering and tracking of goods. Manufacturers and large wholesalers often purchase more robust IMS systems that are designed specifically for their needs.

An inventory management system is responsible for ensuring that the right quantity of the right product is available to customers at the right time and at the right price. A well-designed IMS will help improve efficiency, reduce costs, and minimize the risk of obsolescence or stockouts. Implementing an IMS can be a challenge; many businesses struggle to achieve effective inventory management without a dedicated resource to manage it. However, a solid plan and sound implementation strategies can help to ensure a successful outcome. The first step in implementing a new inventory management system is to understand the benefits and risks of the program.

1.2 PURPOSE

The main purpose of an inventory management system is to help companies track the quantity, location, and condition of all inventory. This information can then be used to make decisions about where to allocate resources and when to order new products. Inventory management systems can also help companies reduce the amount of inventory they have on hand, which can save money and increase profits.

As your business grows, its inventory requirements will grow as well. Your inventory will become more complex, with items coming in from multiple suppliers and multiple warehouses. Managing your inventory manually will be challenging and time-consuming, making it difficult for you to maintain adequate levels of inventory to meet customer demand and grow your business.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

One common problem that existed in most of the systems is the inability to track the inventory in real time. This is because the systems were not integrated with the point-of-sale system. This meant that the inventory was not updated in real time. This resulted in the loss of sales and profits.

2.2 REFERENCES

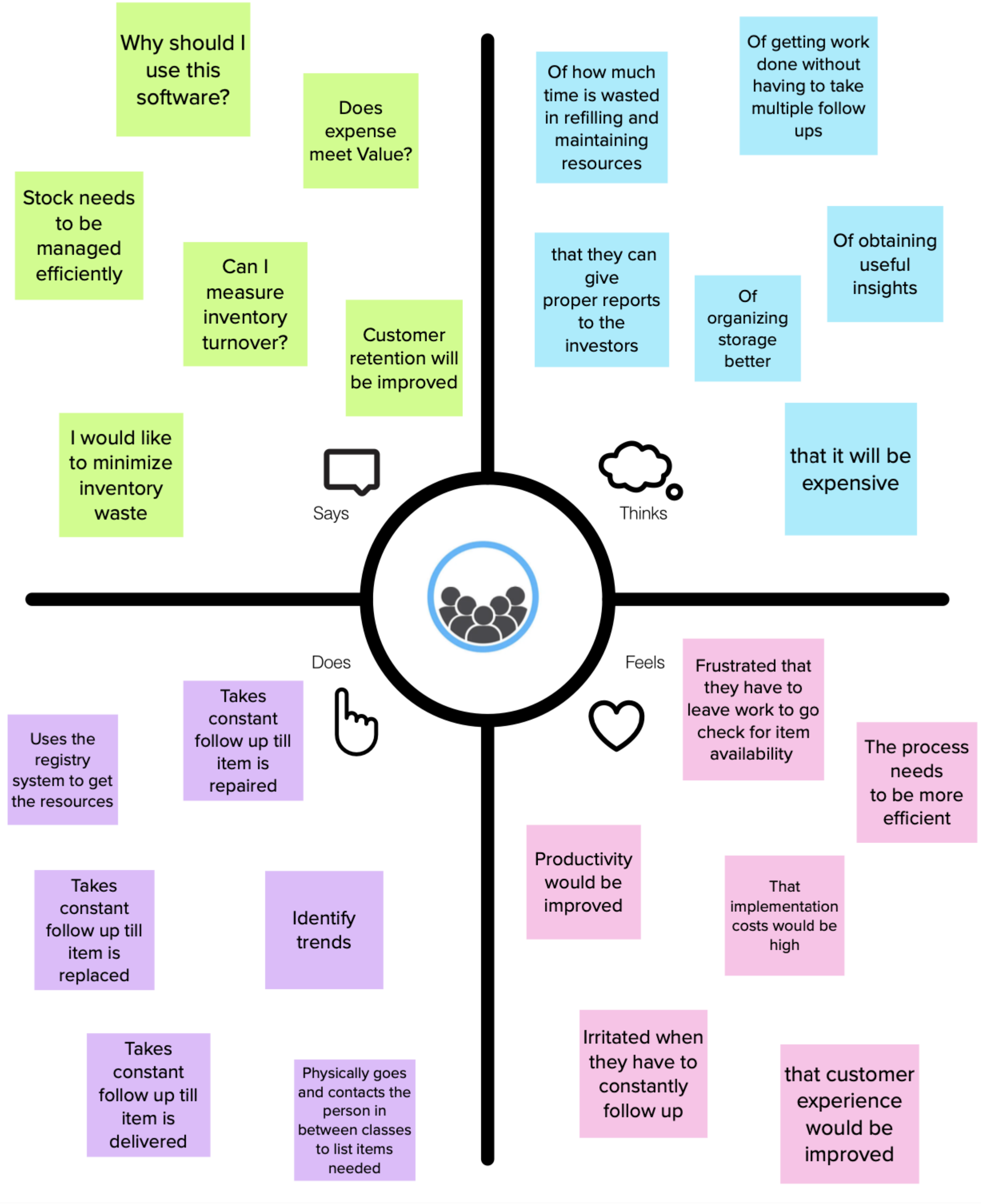
2.3 PROBLEM STATEMENT DEFINITION

The main Objective of this Project is to provide a desktop based application that allows shops to monitor all IMS related information, including stock management, sales data and purchase information. The application enables retailers to manage their products flexibility and have complete insight into what is stored in their inventory, and request additional stock as and when needed.

I am	Retailers and Customers
I'm trying to	Have more insights on stocks and their availability to increase productivity
But	Manual management of the stocks are difficult and existing systems aren't much flexible
Because	Too much stock items cause bigger problems and current systems are obsolete
Which makes me	Want to create better inventory management system and increase the accuracy and flexibility of the vendors

3. IDEATION & PROPOSED SOLUTION

3.1EMPATHY MAP AND CANVAS



Says

Why should I use this software?

Does expense meet Value?

Stock needs to be managed efficiently

Can I measure inventory turnover?

Customer retention will be improved

I would like to minimize inventory waste

Thinks

Of how much time is wasted in refilling and maintaining resources

Of getting work done without having to take multiple follow ups

that they can give proper reports to the investors

Of organizing storage better

Of obtaining useful insights

that it will be expensive

Does

Takes constant follow up till item is repaired

Uses the registry system to get the resources

Takes constant follow up till item is replaced

Identify trends

Takes constant follow up till item is delivered

Physically goes and contacts the person in between classes to list items needed

Feels

Frustrated that they have to leave work to go check for item availability

The process needs to be more efficient

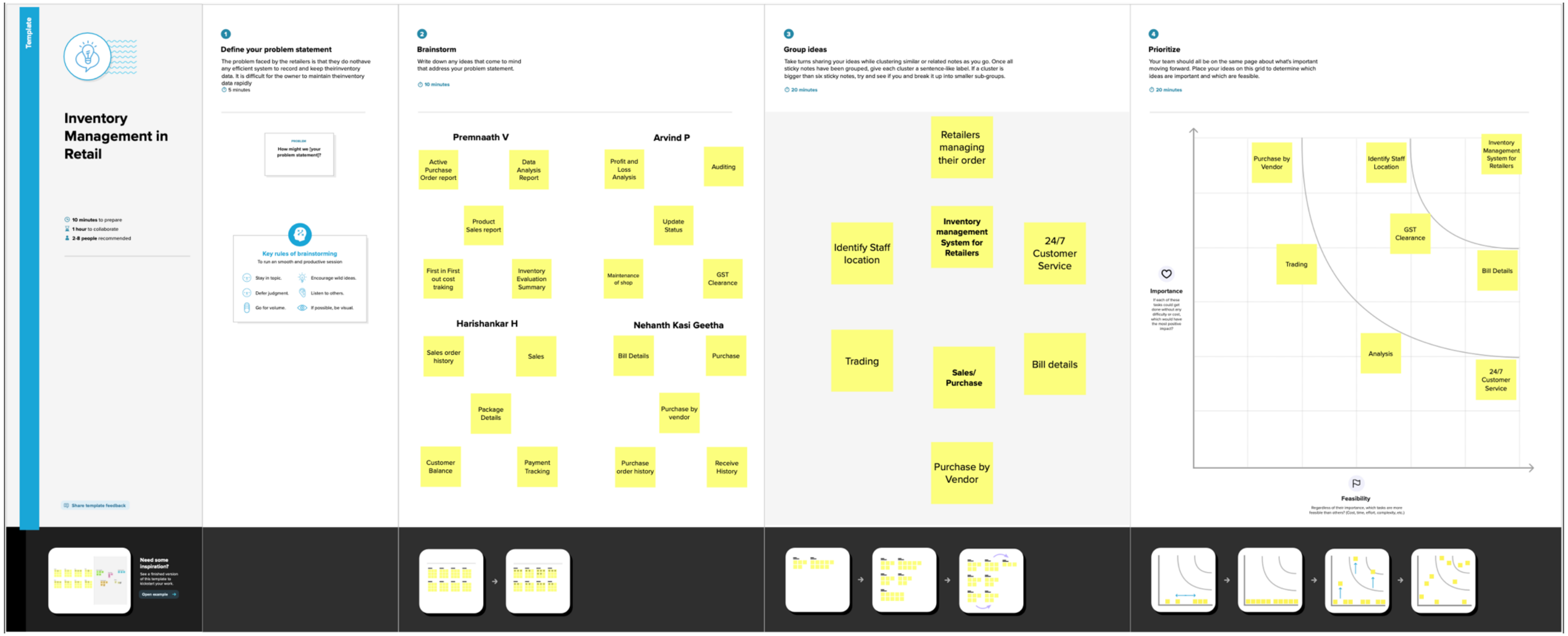
Productivity would be improved

That implementation costs would be high

Irritated when they have to constantly follow up

that customer experience would be improved

3.2 IDEATION & BRAINSTORMING



Template

Inventory Management in Retail

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Share template feedback

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

1 Define your problem statement

The problem faced by the retailers is that they do nothave any efficient system to record and keep theirinventory data. It is difficult for the owner to maintain theinventory data rapidly

5 minutes

PROBLEM
How might we [your problem statement]?

Key rules of brainstorming
To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2 Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

4 Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

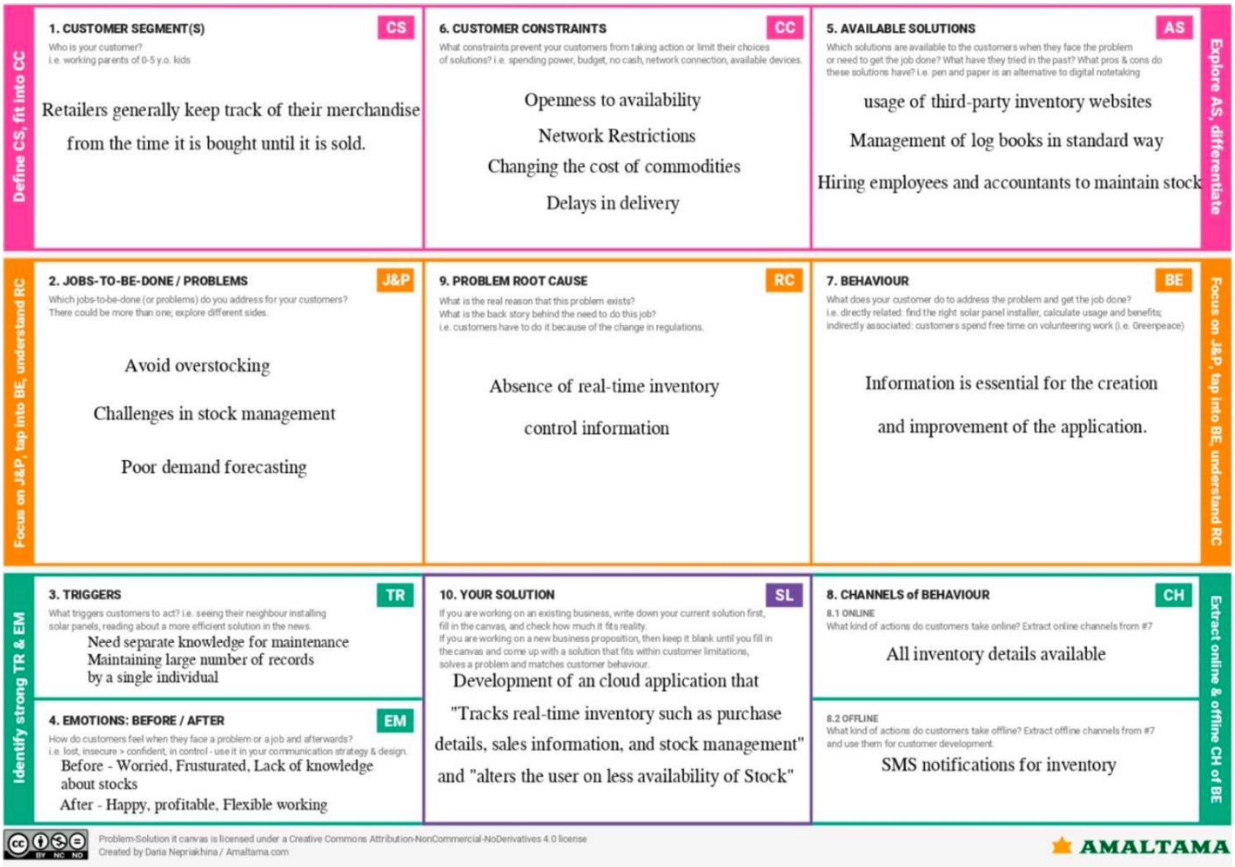
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

3.3PROPOSED SOLUTION

1	Problem Statement (Problem to be solved)	The problem statement aims to make desktop application for retailers and to track all areas of Inventory Management System like purchase details, sales details, stock management and other policies.
2	Idea / Solution Description	The application is developed to help retailers track and manage stocks related to their own products. The System will ask the retailers to create their accounts by providing essential details. Retailers can access their accounts by logging into the application. Once retailers successfully log in to the application they can update their inventory details, also users will be able to add new stock by submitting essential details related to the stock. They can view details of the current inventory. The System will automatically send an email alert to the retailers if there is no stock found in their accounts. So that they can order new stock.
3	Novelty / Uniqueness	Apart from the standard features of the inventory management system like handling products, warehouses, locations we also plan to include the feature of sales prediction using
		regression and the previous sales data within our application. We also make the development and maintenance easier by containerizing the app using Docker
4	Social Impact / Customer Satisfaction	With this system we aim to make better use of the inventory available for the retailers. This improves the management and reduces excess inventory and thus reduces the wastage of products. It also improves the relationship with vendors and suppliers and can negotiate better deals with the suppliers by knowing the demand beforehand.
5	Business Model (Revenue Satisfaction)	Retailers can order the right amount and type of stock at the right time with the aid of an inventory

		management system. It eliminates the unnecessary expense for the retailers.
6	Scalability of the Solution	A scalable cloud architecture is made possible through virtualization. Unlike physical machines whose resources and performance are relatively set, virtual machines virtual machines (VMs) that we use in IBM cloud are highly flexible and can be easily scaled up or down. Kubernetes allows users to horizontally scale the total containers used based on the application requirements, which may change over time. It's easy to change the number via the command line

3.4 PROBLEM SOULTION FIT



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Retailers generally keep track of their merchandise from the time it is bought until it is sold.

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

Openness to availability

Network Restrictions

Changing the cost of commodities

Delays in delivery

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

usage of third-party inventory websites

Management of log books in standard way

Hiring employees and accountants to maintain stock

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

Avoid overstocking

Challenges in stock management

Poor demand forecasting

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

Absence of real-time inventory control information

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

Information is essential for the creation and improvement of the application.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.

Need separate knowledge for maintenance
Maintaining large number of records
by a single individual

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

Before - Worried, Frusturated, Lack of knowledge about stocks

After - Happy, profitable, Flexible working

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.

Development of an cloud application that "Tracks real-time inventory such as purchase details, sales information, and stock management" and "alters the user on less availability of Stock"

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

All inventory details available

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

SMS notifications for inventory

Extract online & offline CH of BE

AMALTAMA

4. REQUIREMENT ANALYSIS

4.1 SOLUTION & TECHNICAL REQUIREMENTS

Functional Requirements

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registering through a form Registering through mail
FR-2	User Confirmation	Email confirmation OTP confirmation
FR-3	Login	Log in to the application by entering required credentials (email ID and password)
FR-4	Dashboard	View the products details (Name, quantity)
FR-5	Add items to the Inventory list	Users can add items that they wish to buy to the inventory
FR-6	Stock Updation	Increasing the availability of a particular product

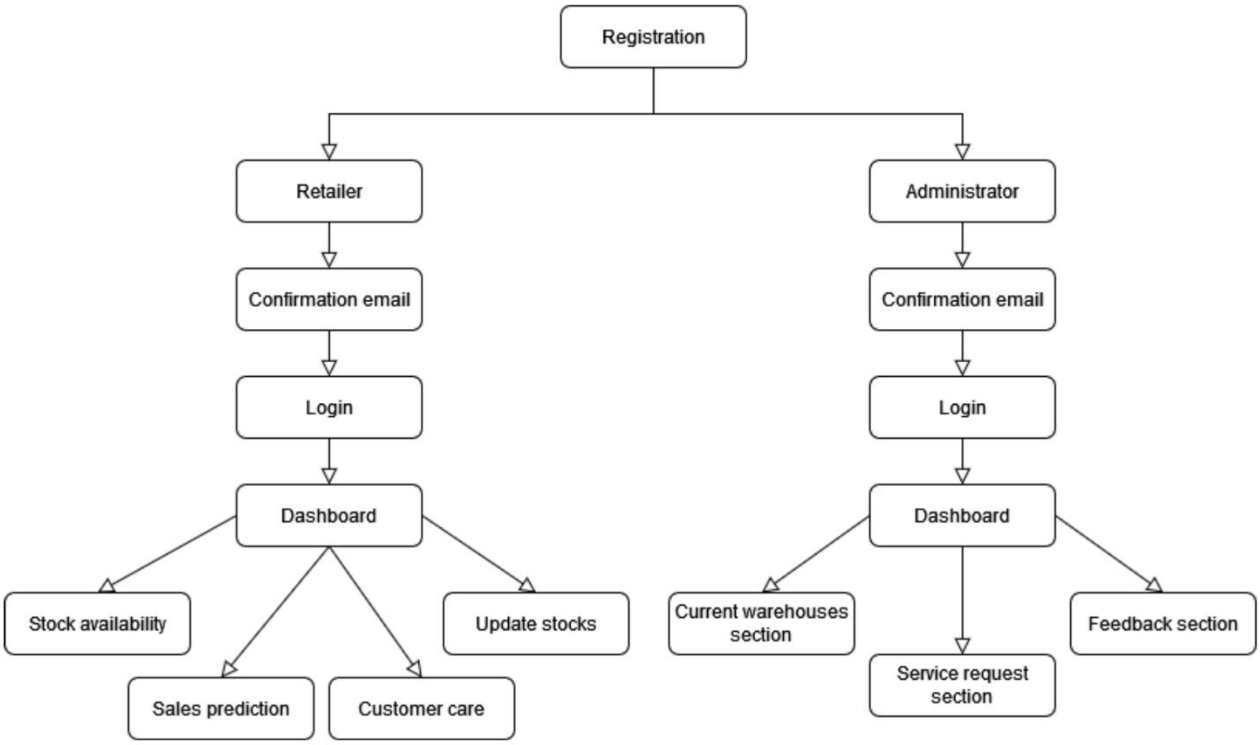
Non-Functional Requirements

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	If the system has a steep learning curve, then it would mostly not be purchased by the company needing an inventory management system. • The UI is simple and easy to navigate • Consistent design and colours are used. • The webpages are responsive • Email delivery is to be fast
NFR-2	**Security**	Security refers to the safety and management of the inventory of a company such that only authorised personnel are allowed to access them. • Login system is used to provide authentication. • Users need to create account and verify it with their email OTP. • Cookie based security is user for authentication on client side.

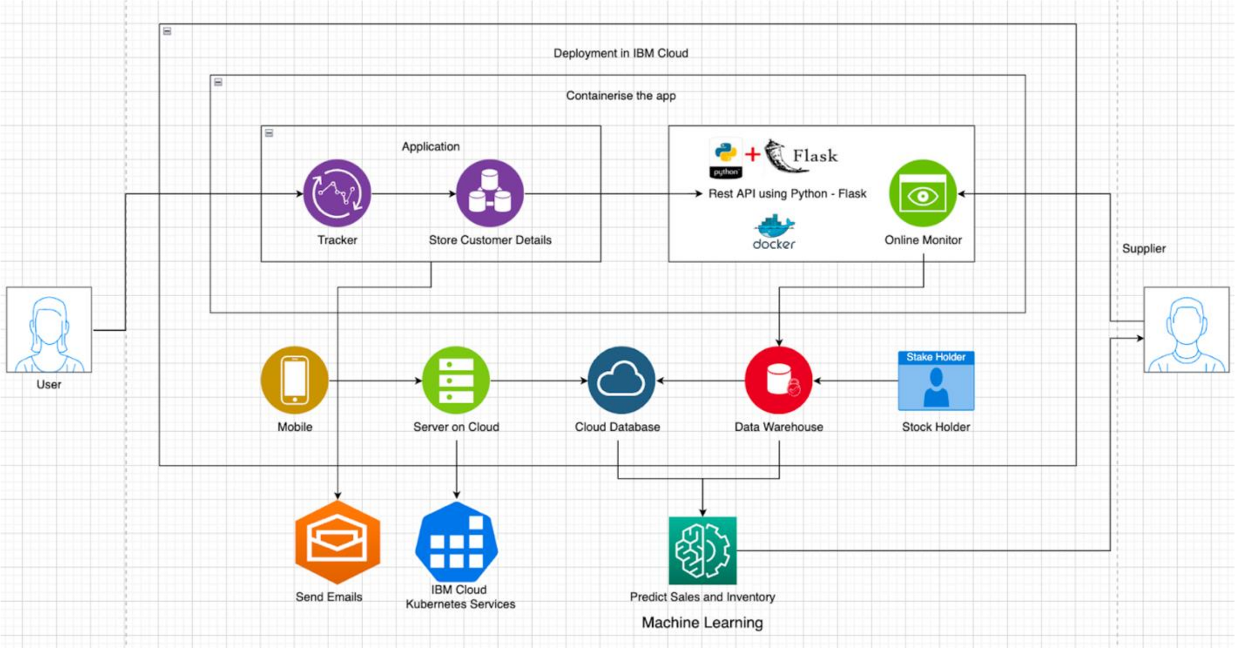
NFR-3	**Reliability**	• Exception handling will be done at the code level to ensure that the app performs well even when errors happen in the runtime • Multiple instances of the App would be online to ensure continued operation
NFR-4	**Performance**	Performance of an inventory management system depends on the efficiency with which various tasks in it can be executed. • Reduces manpower, cost and saves time. Emails will be sent automatically when stocks are not available. • Makes the business process more efficient. • Improves organizations performance. • It will be perform fast and secure even at the lower bandwidth
NFR-5	**Availability**	The use of IBM DB2 ensures high availability
NFR-6	**Scalability**	The scalability of an inventory management system refers to the extensibility of its operations. • DB2 is highly Scalable • The code is developed efficiently to easily add new features without many changes by reusing the code. • Docker in IBM Container registry is used which is highly scalable

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 SOLUTION &TECHNICAL ARCHITECTURE



5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I can register for the application through E-mail.	I can access my account / dashboard	Medium	Sprint-1
	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application.	I can get a confirmation for my email and password and create an authenticated account.	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering the registered email & password.	I can log onto the application with the verified email and password	High	Sprint-1
	Dashboard	USN-5	As a user, I can view the products which are available.	Once I log on to the application, I can view the inventory.	High	Sprint-2

	Stock Update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list.	If any of the products are not available, as a user I can update the inventory.	Medium	Sprint-2
	Sales Prediction	USN-7	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	The sales prediction tool should forecast the sales so that I, as a User, can order appropriately.	Medium	Sprint-3
Administrator	Request to Customer Care	USN-8	As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.	As a user, I can contact Customer Care and get support from them.	Low	Sprint-4
	Give feedback	USN-9	I should be able to report any difficulties I experience as a report.	As user, I can give my support in my possible ways to the administrator and to the administration.	Medium	Sprint-4

6. PROJECT PLANNING & SCHEDULING

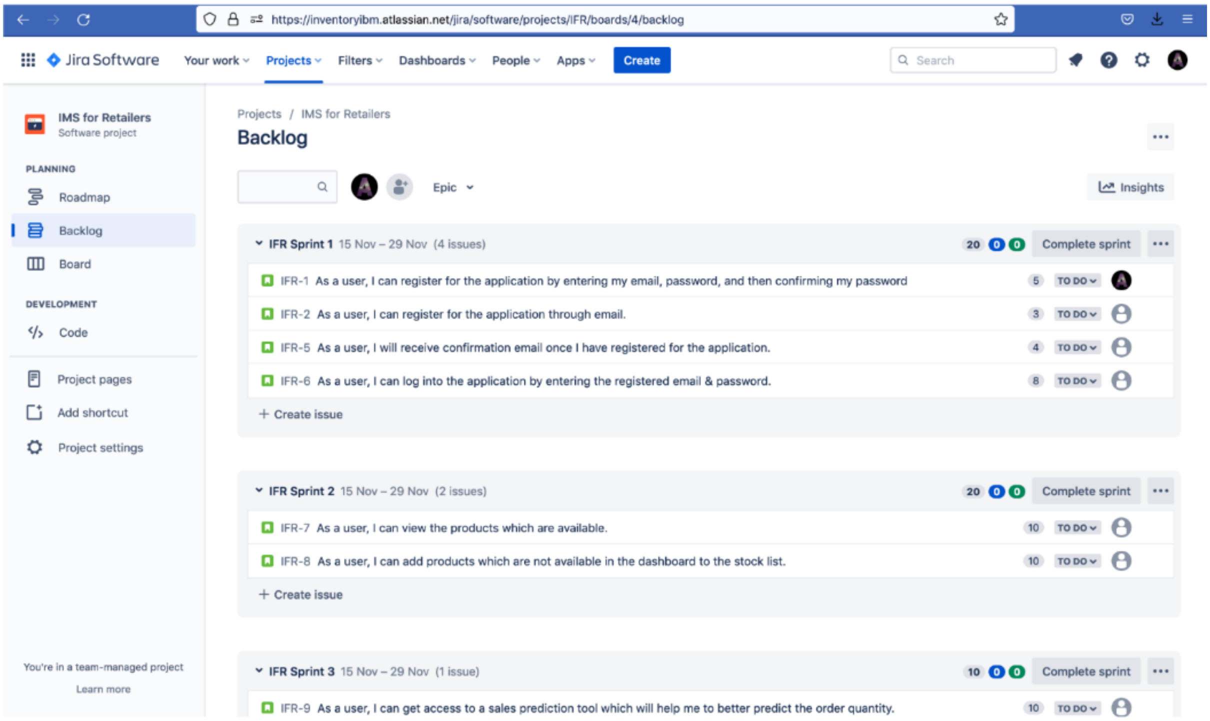
6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and then confirming my password.	5	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1		USN-2	As a user, I can register for the application through email.	3	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1	Confirmation	USN-3	As a user, I will receive confirmation email once I have registered for the application.	4	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-1	Login	USN-4	As a user, I can log into the application by entering the registered email & password	8	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-2	Dashboard	USN-5	As a user, I can view the products which are available.	10	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-2	Stock Update	USN-6	As a user, I can add products which are not available in the dashboard to the stock list.	10	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-3	Sales Prediction	USN-7	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	10	High	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-4	Administration	USN-8	As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.	10	Low	Harishankar. H, Arvind P, Nehanth K G, Premnaath V
Sprint-4		USN-9	I should be able to report any difficulties I experience to the administrator.	10	Medium	Harishankar. H, Arvind P, Nehanth K G, Premnaath V

6.2 SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	4 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	15 Nov 2022
Sprint-3	10	6 Days	07 Nov 2022	12 Nov 2022	10	22 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	25 Nov 2022

6.3 REPORTS FROM JIRA



Projects / IMS for Retailers

Backlog

IFR Sprint 1 15 Nov – 29 Nov (4 issues) 20 0 0

IFR-1	As a user, I can register for the application by entering my email, password, and then confirming my password	5	TO DO
IFR-2	As a user, I can register for the application through email.	3	TO DO
IFR-5	As a user, I will receive confirmation email once I have registered for the application.	4	TO DO
IFR-6	As a user, I can log into the application by entering the registered email & password.	8	TO DO

+ Create issue

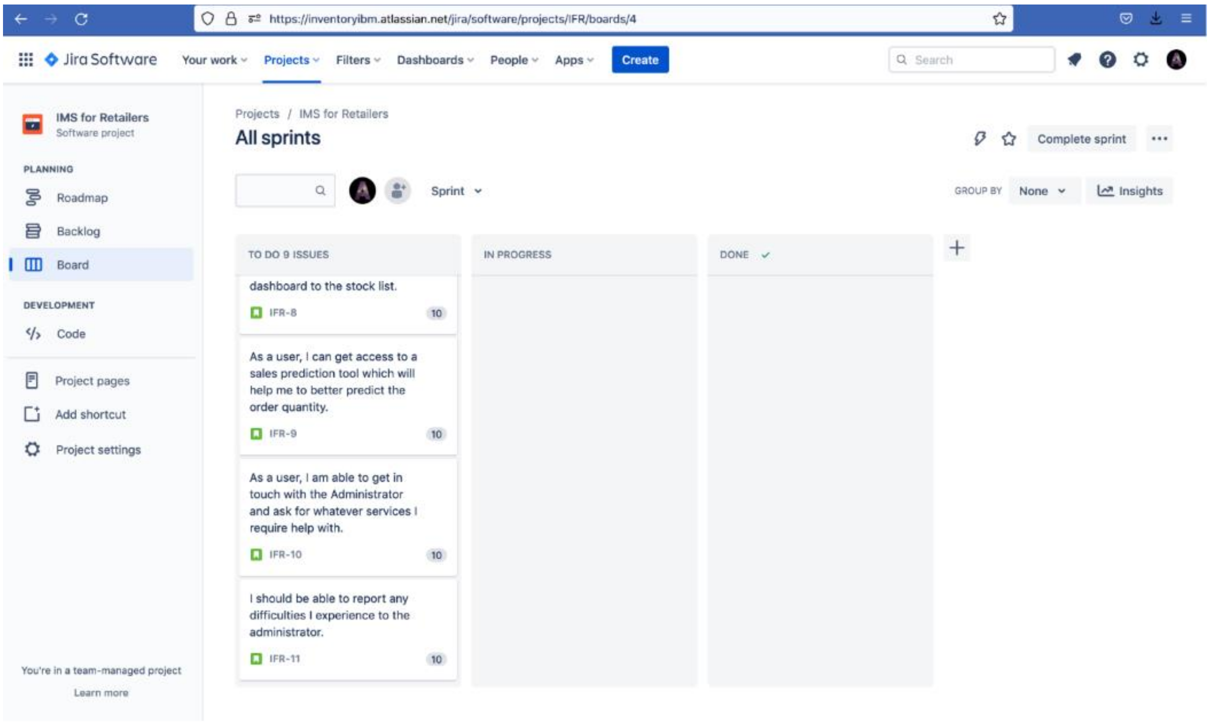
IFR Sprint 2 15 Nov – 29 Nov (2 issues) 20 0 0

IFR-7	As a user, I can view the products which are available.	10	TO DO
IFR-8	As a user, I can add products which are not available in the dashboard to the stock list.	10	TO DO

+ Create issue

IFR Sprint 3 15 Nov – 29 Nov (1 issue) 10 0 0

IFR-9	As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.	10	TO DO



Projects / IMS for Retailers

All sprints

TO DO 9 ISSUES

IN PROGRESS

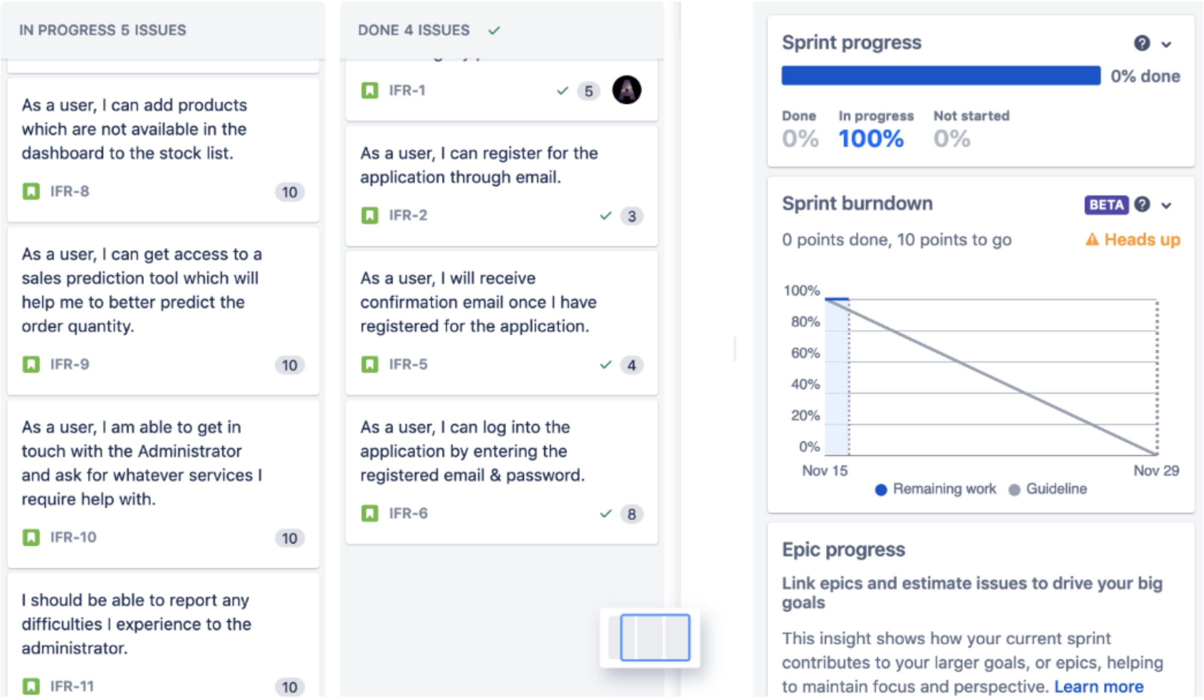
DONE ✓

dashboard to the stock list.
IFR-8 10

As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.
IFR-9 10

As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.
IFR-10 10

I should be able to report any difficulties I experience to the administrator.
IFR-11 10



IN PROGRESS 5 ISSUES

As a user, I can add products which are not available in the dashboard to the stock list.

IFR-8 10

As a user, I can get access to a sales prediction tool which will help me to better predict the order quantity.

IFR-9 10

As a user, I am able to get in touch with the Administrator and ask for whatever services I require help with.

IFR-10 10

I should be able to report any difficulties I experience to the administrator.

IFR-11 10

DONE 4 ISSUES ✓

IFR-1 ✓ 5

As a user, I can register for the application through email.

IFR-2 ✓ 3

As a user, I will receive confirmation email once I have registered for the application.

IFR-5 ✓ 4

As a user, I can log into the application by entering the registered email & password.

IFR-6 ✓ 8

Sprint progress

0% done

Done 0%

In progress 100%

Not started 0%

Sprint burndown BETA

0 points done, 10 points to go

⚠ Heads up

Epic progress

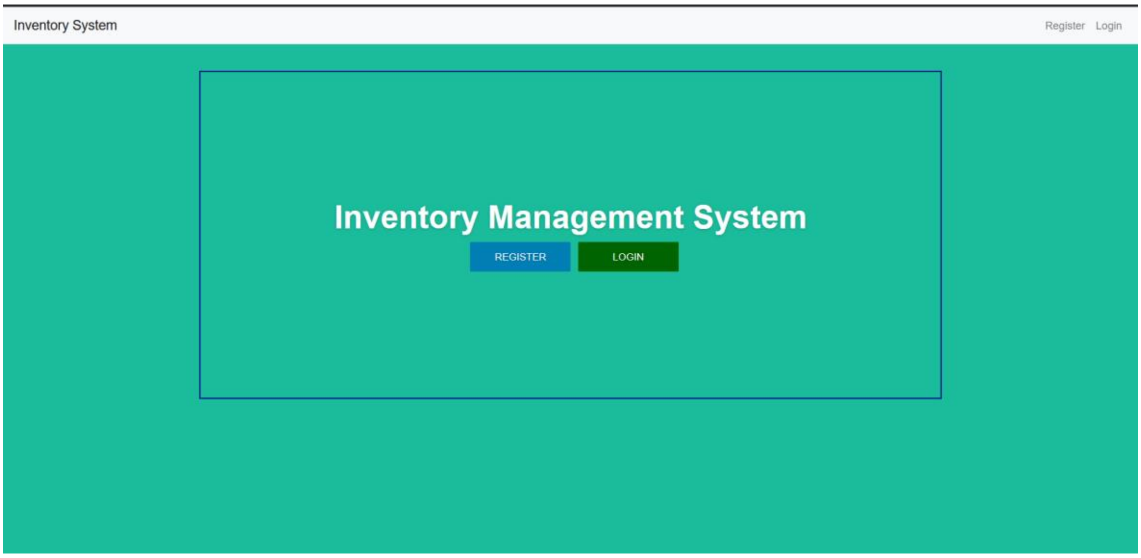
Link epics and estimate issues to drive your big goals

This insight shows how your current sprint contributes to your larger goals, or epics, helping to maintain focus and perspective. Learn more

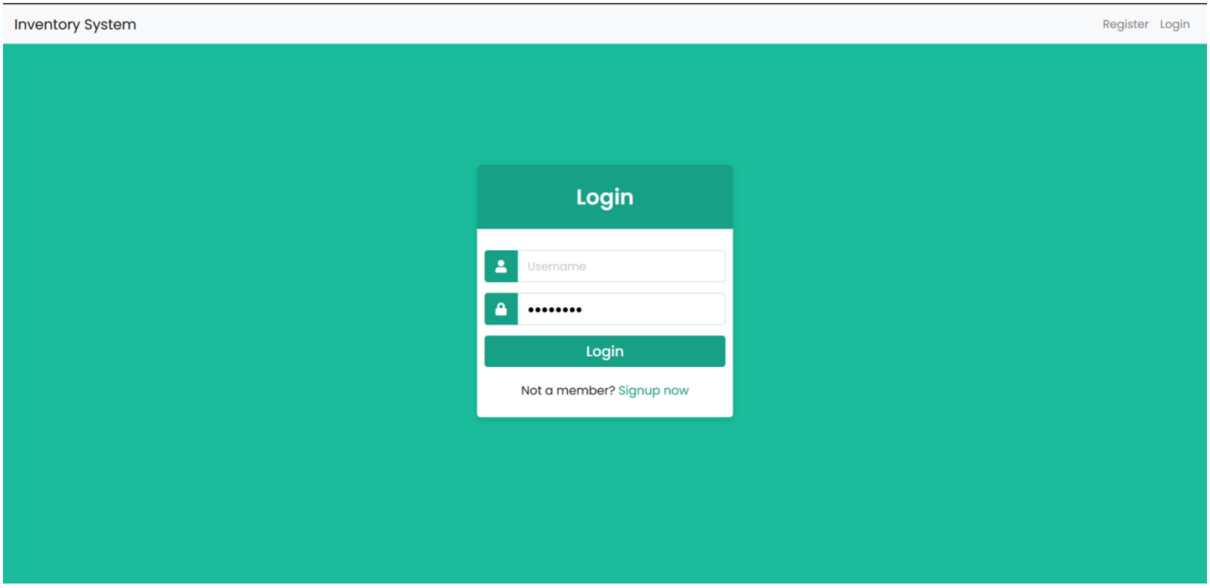
7. CODING & SOLUTIONING (FEATURES ADDED ALONG WITH CODE)

7.1 FEATURE 1

Used flask web framework to create an interactive dashboard



Users can register or login through this dashboard



7.2 FEATURE 2

Used SendGrid for autonomous emails

7.3 DATABASE SCHEMA (IF APPLICABLE)

8. TESTING

8.1TEST CASES

8.2USER ACCEPTANCE TESTING

9. RESULTS

9.1PERFORMANCES METRICS

10. ADVANTAGES & DISADVANTAGES

10.1 ADVANTAGES

- Used for small organization
- Low stock alert as email

10.2 DISADVANTAGES

- This application is not suitable for those organization where there is large quantity
of product and different level of warehouses.
- This software application is able to generate only simple reports.
- Single admin panel is only made.
- It is not suitable for large organization.

11. CONCLUSION

To conclude, Inventory Management System for retailers is a simple web-based application suitable for SMEs. It has all the necessities of a basic Inventory management system which are then used by organizations. Our team is successful in making the application where we can update, insert and delete the item as per the requirement. This application also sends an email alert when stock inventory is low. Though it has some limitations, our team strongly believes that the implementation of this system will surely benefit the organization.

12. FUTURE SCOPE

Since this project was started with very little knowledge about the Inventory Management System, we came to know about the enhancement capability during the process of development. Some of the features which we can implement for the betterment and effectiveness of our project are listed below:

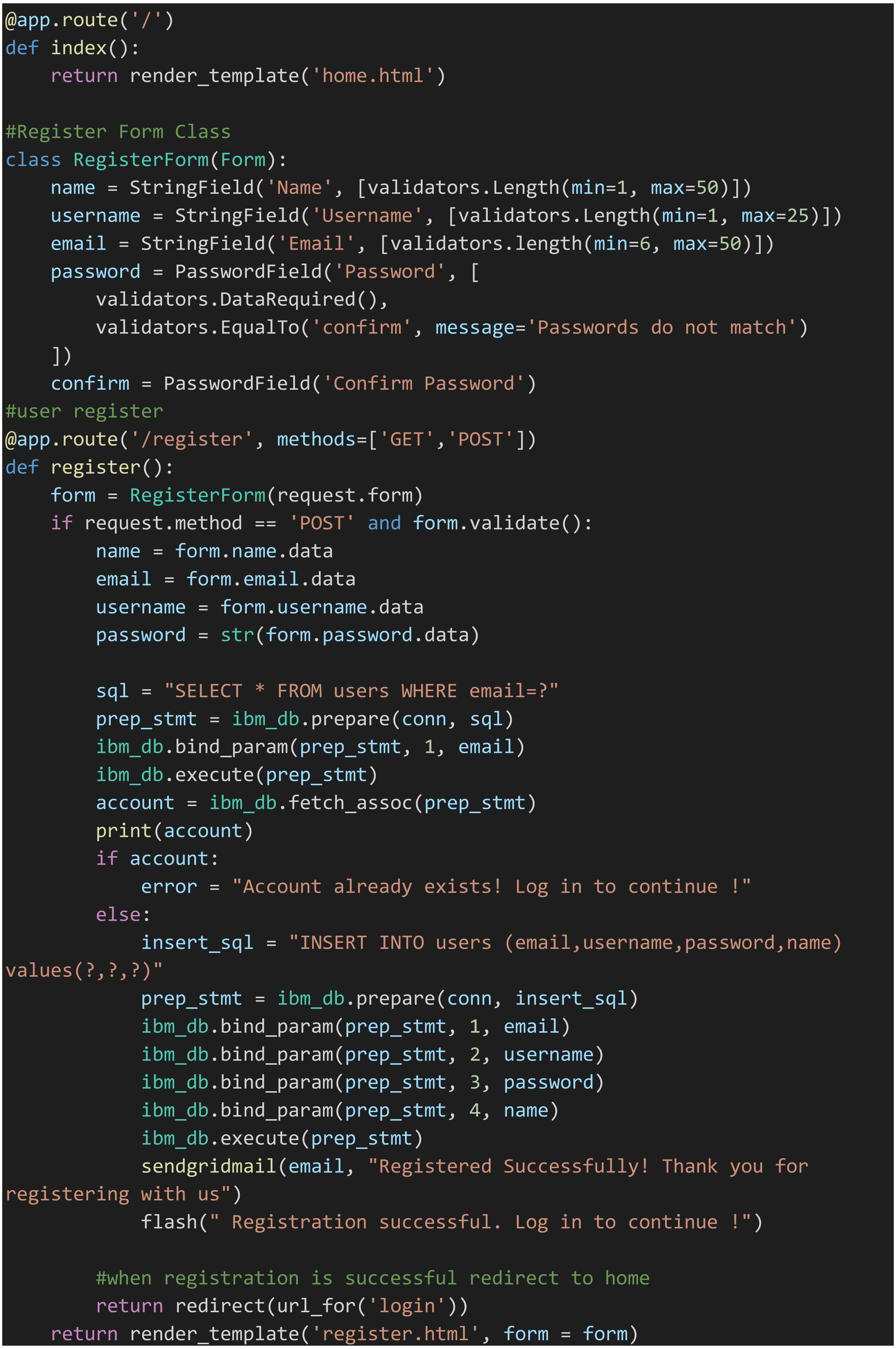
- Interactive user interface design.
- Manage Stock Godown wise.

13. APPENDIX

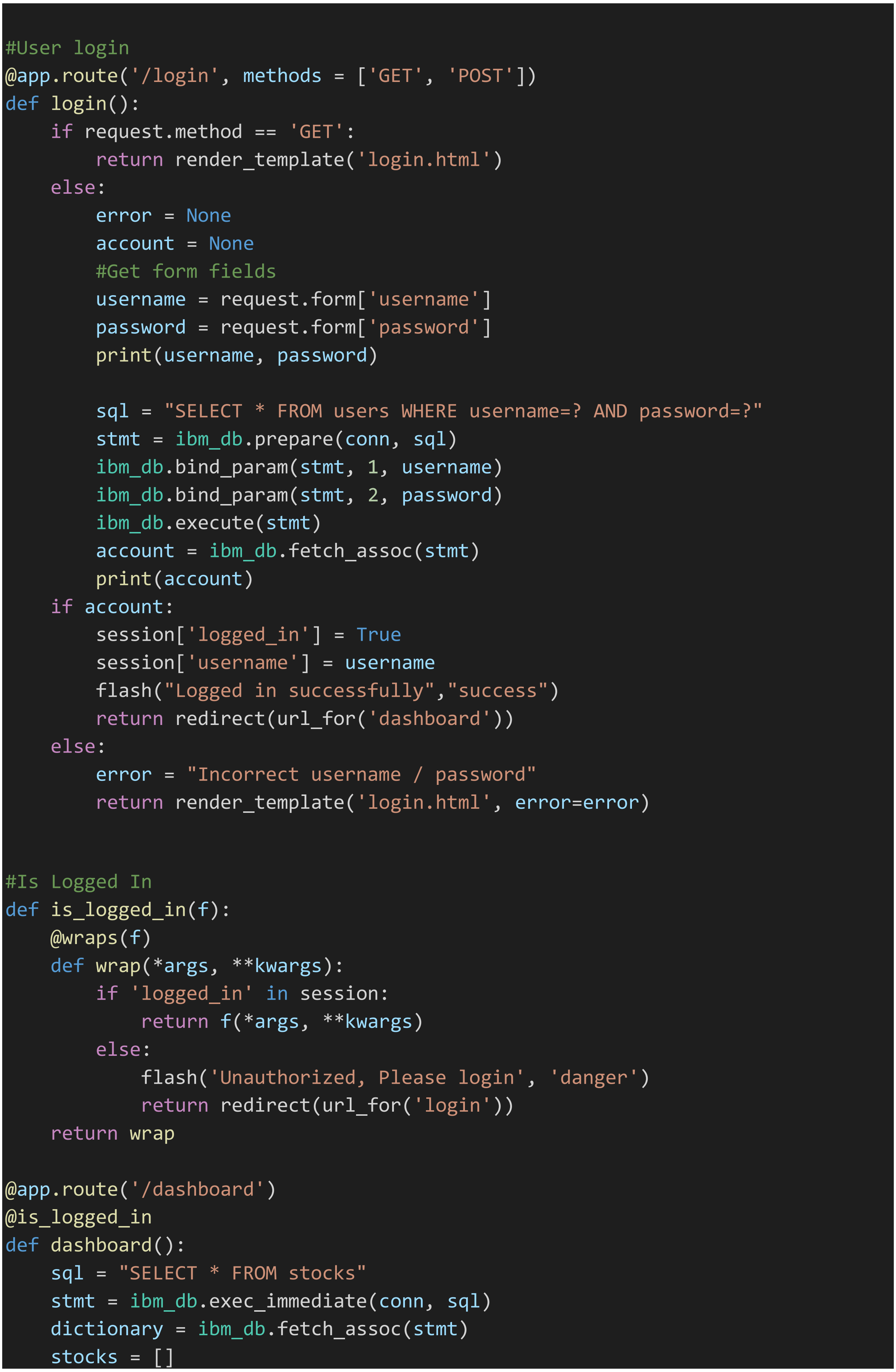
13.1 SOURCE CODE

app.py

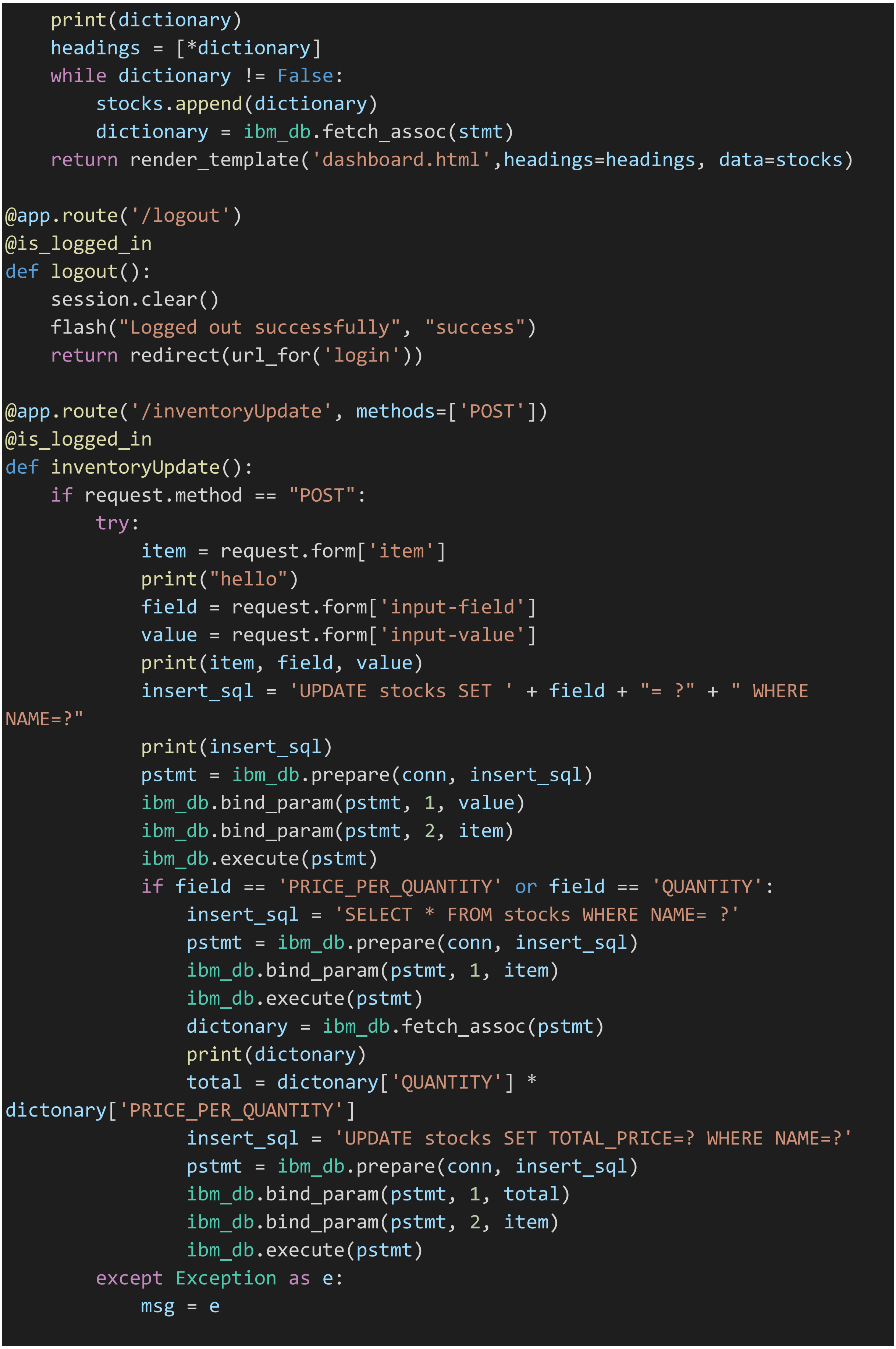
```python
from flask import Flask, render_template, flash, redirect, url_for, session, request, logging
from wtforms import Form, StringField, TextAreaField, PasswordField, validators, SelectField, IntegerField
import ibm_db
from functools import wraps
from datetime import datetime, timedelta
import sendgrid
import os
from sendgrid.helpers.mail import Mail, Email, To, Content


app = Flask(__name__)
app.secret_key = 'kekcwcekqwodq'
#IBM DB2 Connection
try:
    conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=31249;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=cqg39702;PWD=hIRRyoYSNHJxjqQq", "", "")
except:
    print("Unable to connect: ", ibm_db.conn_error())


def sendgridmail(user,TEXT):
    sg = sendgrid.SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
    from_email = Email(os.environ.get('SENDGRID_FROM_EMAIL'))
    to_email = To(user)
    subject = "Registered Successfully"
    content = Content("text/plain",TEXT)
    mail = Mail(from_email, to_email, subject, content)

    # Get a JSON-ready representation of the Mail object
    mail_json = mail.get()
    # Send an HTTP POST request to /mail/send
    response = sg.client.mail.send.post(request_body=mail_json)
    print(response.status_code)
    print(response.headers)
```



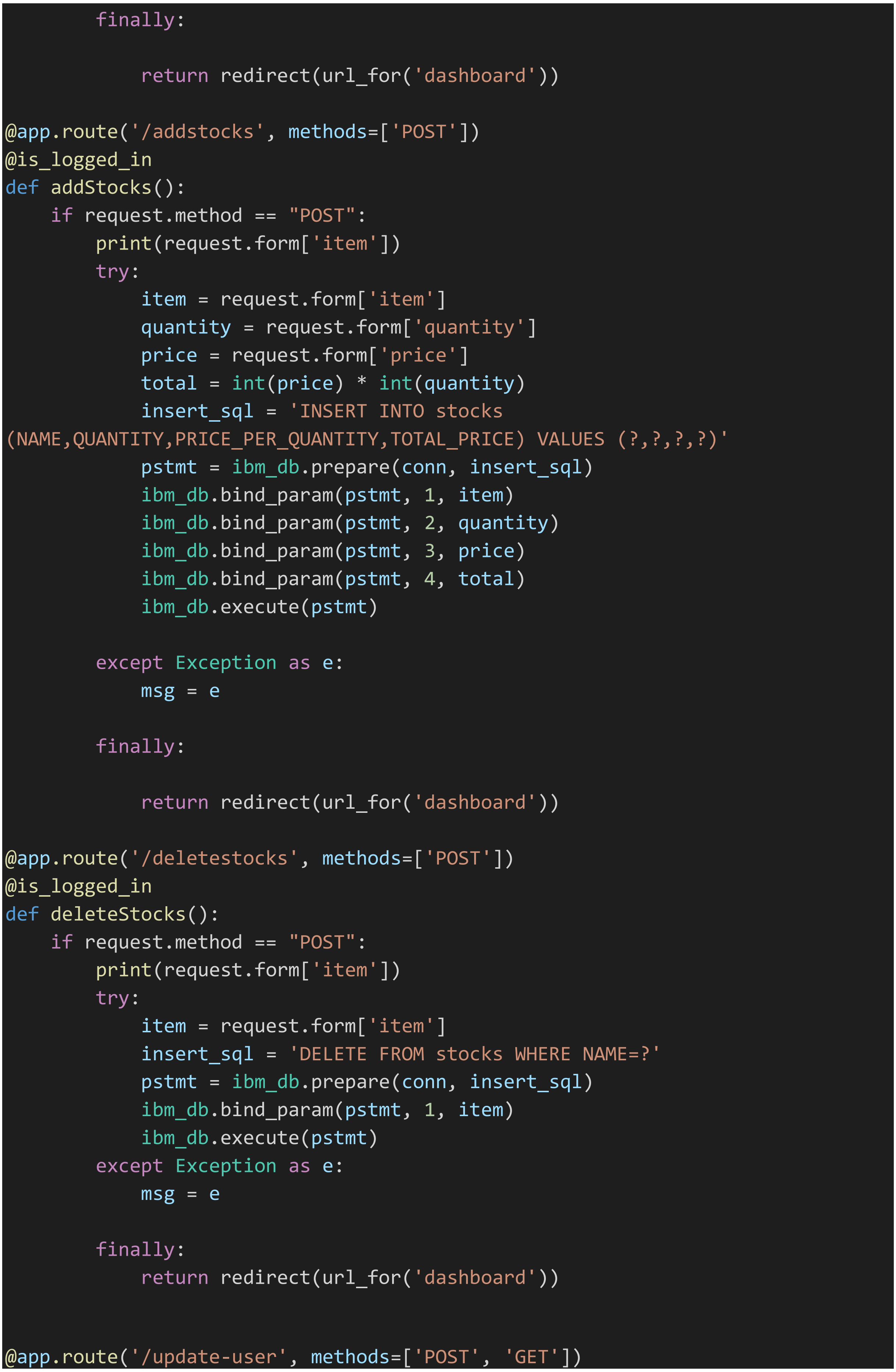
```
@app.route('/')
def index():
    return render_template('home.html')

#Register Form Class
class RegisterForm(Form):
    name = StringField('Name', [validators.Length(min=1, max=50)])
    username = StringField('Username', [validators.Length(min=1, max=25)])
    email = StringField('Email', [validators.length(min=6, max=50)])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords do not match')
    ])
    confirm = PasswordField('Confirm Password')
#user register
@app.route('/register', methods=['GET','POST'])
def register():
    form = RegisterForm(request.form)
    if request.method == 'POST' and form.validate():
        name = form.name.data
        email = form.email.data
        username = form.username.data
        password = str(form.password.data)

        sql = "SELECT * FROM users WHERE email=?"
        prep_stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(prep_stmt, 1, email)
        ibm_db.execute(prep_stmt)
        account = ibm_db.fetch_assoc(prep_stmt)
        print(account)
        if account:
            error = "Account already exists! Log in to continue !"
        else:
            insert_sql = "INSERT INTO users (email,username,password,name)
values(?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, email)
            ibm_db.bind_param(prep_stmt, 2, username)
            ibm_db.bind_param(prep_stmt, 3, password)
            ibm_db.bind_param(prep_stmt, 4, name)
            ibm_db.execute(prep_stmt)
            sendgridmail(email, "Registered Successfully! Thank you for
registering with us")
            flash(" Registration successful. Log in to continue !")

        #when registration is successful redirect to home
        return redirect(url_for('login'))
    return render_template('register.html', form = form)
```



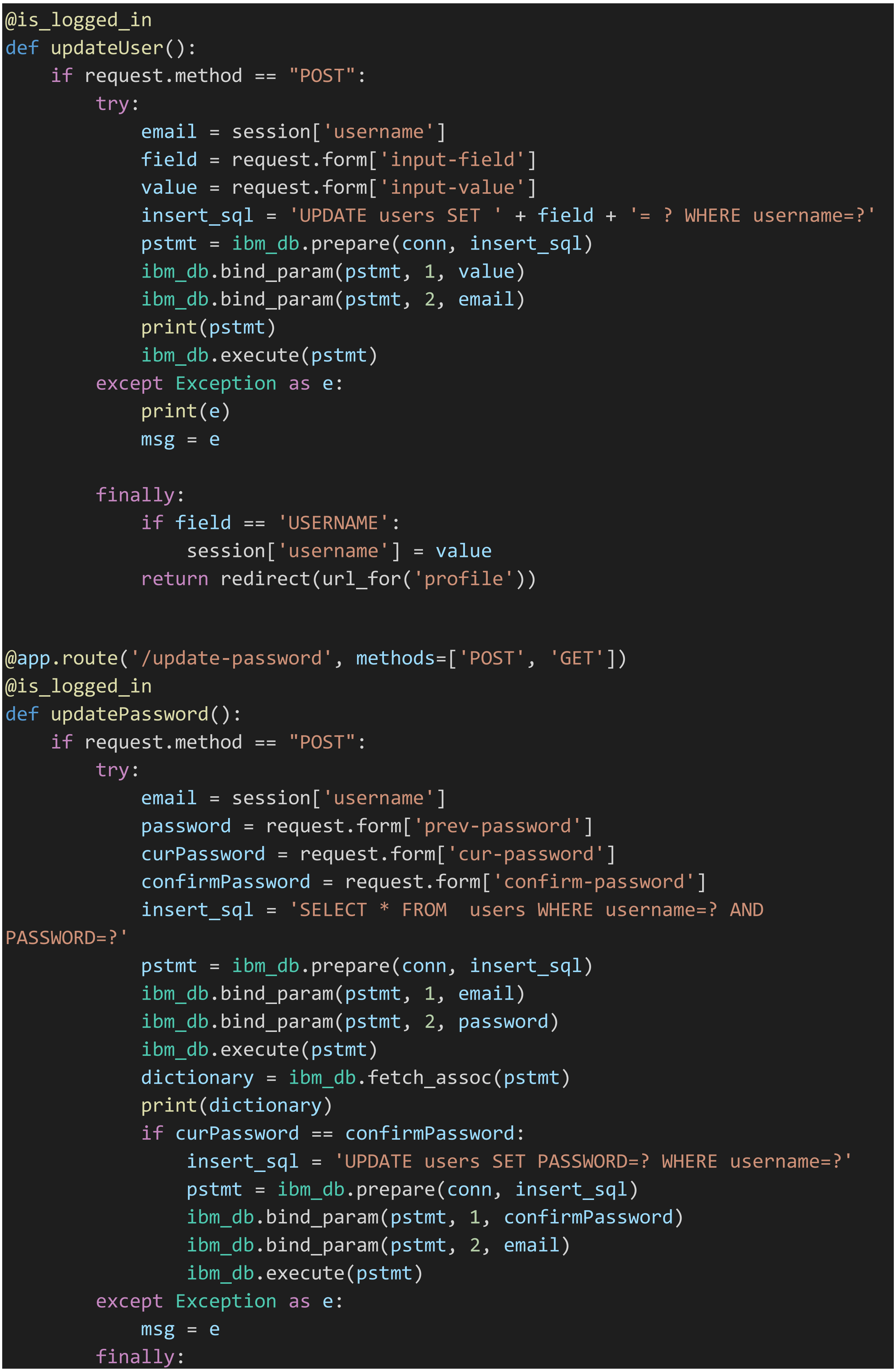
```python
#User login
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html')
    else:
        error = None
        account = None
        #Get form fields
        username = request.form['username']
        password = request.form['password']
        print(username, password)

        sql = "SELECT * FROM users WHERE username=? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
    if account:
        session['logged_in'] = True
        session['username'] = username
        flash("Logged in successfully","success")
        return redirect(url_for('dashboard'))
    else:
        error = "Incorrect username / password"
        return render_template('login.html', error=error)


#Is Logged In
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return f(*args, **kwargs)
        else:
            flash('Unauthorized, Please login', 'danger')
            return redirect(url_for('login'))
    return wrap

@app.route('/dashboard')
@is_logged_in
def dashboard():
    sql = "SELECT * FROM stocks"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    stocks = []
```



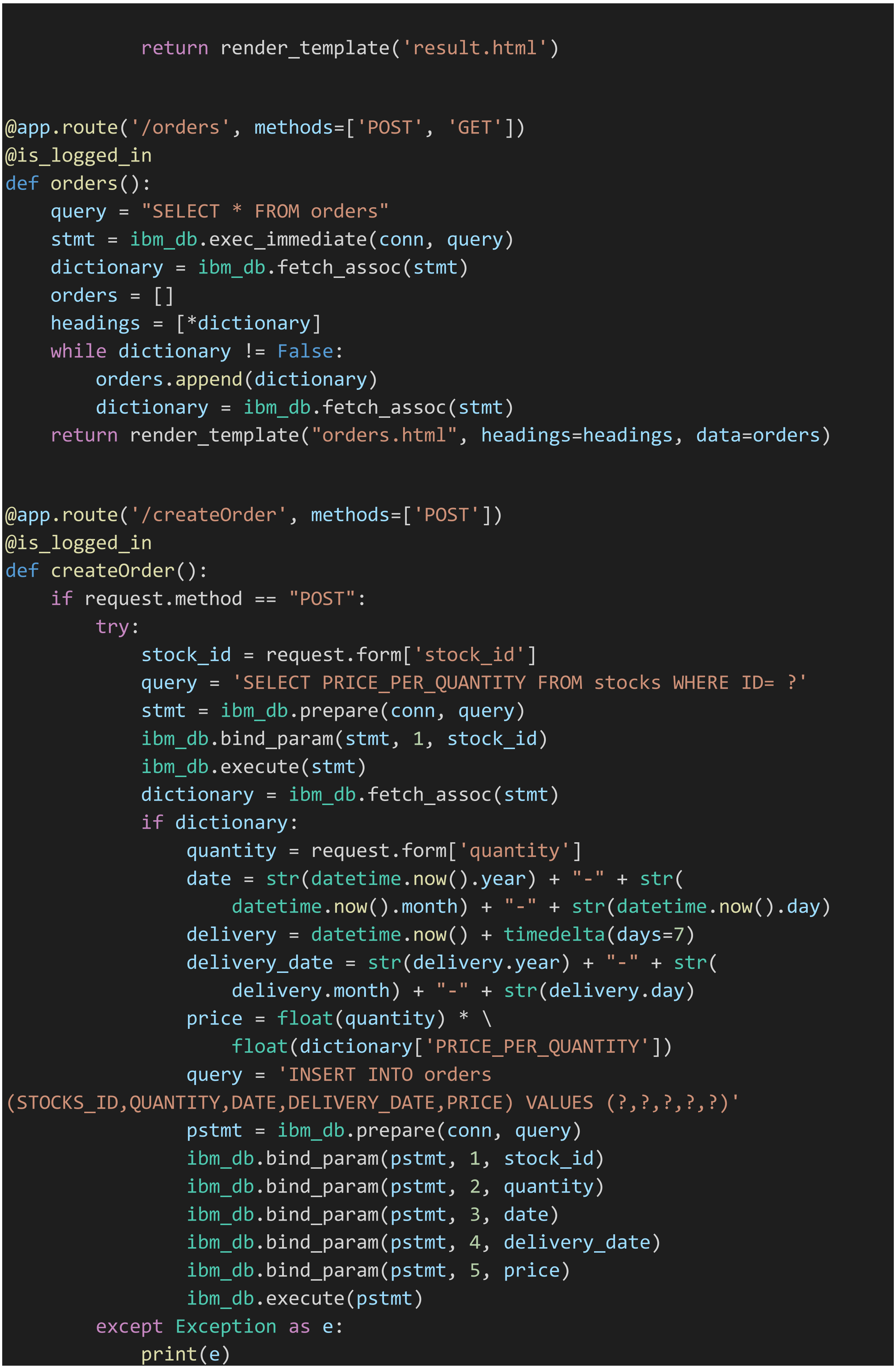
```
    print(dictionary)
    headings = [*dictionary]
    while dictionary != False:
        stocks.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template('dashboard.html',headings=headings, data=stocks)

@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash("Logged out successfully", "success")
    return redirect(url_for('login'))

@app.route('/inventoryUpdate', methods=['POST'])
@is_logged_in
def inventoryUpdate():
    if request.method == "POST":
        try:
            item = request.form['item']
            print("hello")
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE stocks SET ' + field + "= ?" + " WHERE
NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
            if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
                insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, item)
                ibm_db.execute(pstmt)
                dictonary = ibm_db.fetch_assoc(pstmt)
                print(dictonary)
                total = dictonary['QUANTITY'] *
dictonary['PRICE_PER_QUANTITY']
                insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, total)
                ibm_db.bind_param(pstmt, 2, item)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
```



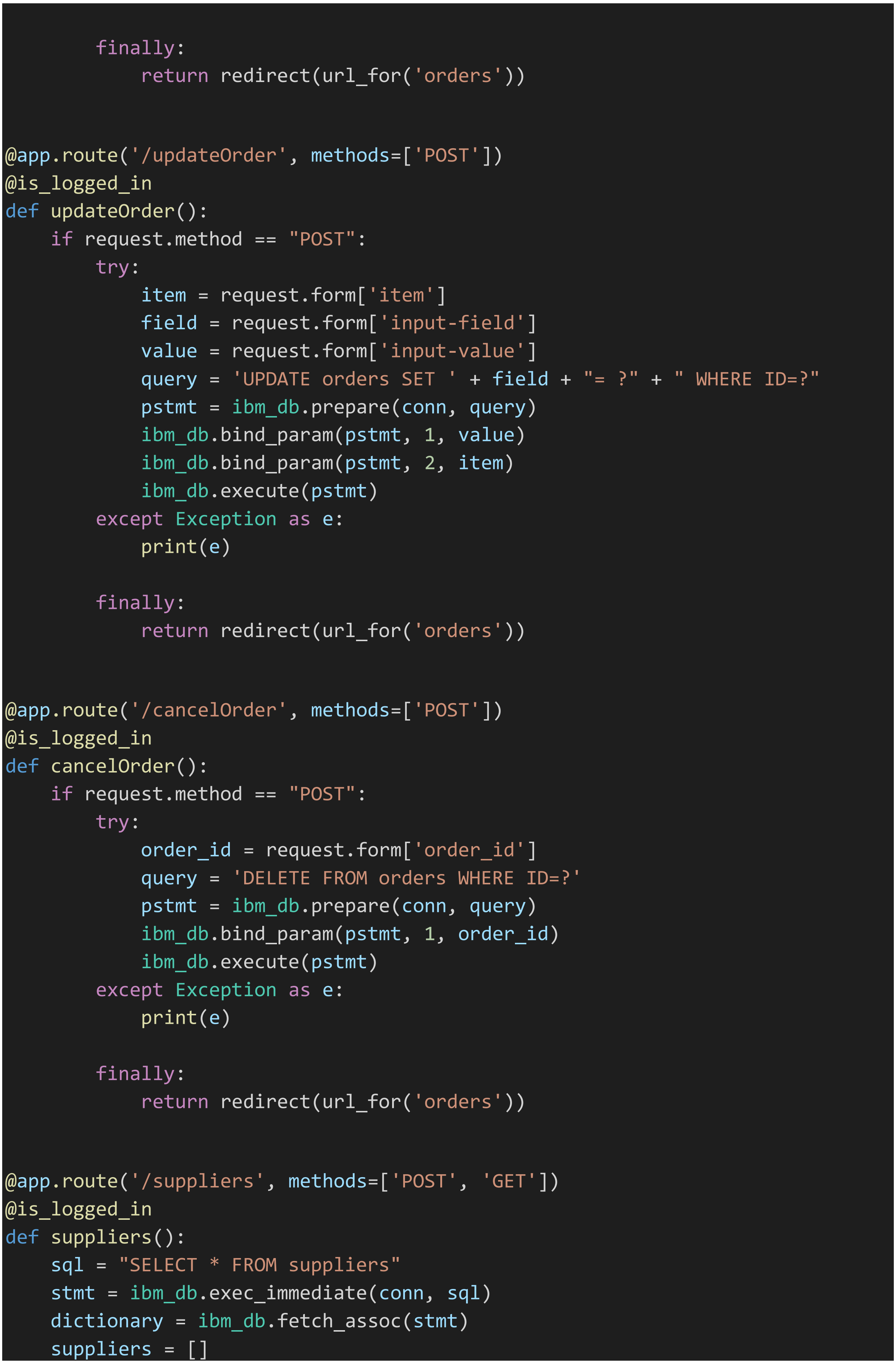
```
        finally:

            return redirect(url_for('dashboard'))

@app.route('/addstocks', methods=['POST'])
@is_logged_in
def addStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            quantity = request.form['quantity']
            price = request.form['price']
            total = int(price) * int(quantity)
            insert_sql = 'INSERT INTO stocks
(NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES (?,?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, price)
            ibm_db.bind_param(pstmt, 4, total)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:

            return redirect(url_for('dashboard'))

@app.route('/deletestocks', methods=['POST'])
@is_logged_in
def deleteStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            insert_sql = 'DELETE FROM stocks WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('dashboard'))


@app.route('/update-user', methods=['POST', 'GET'])
```



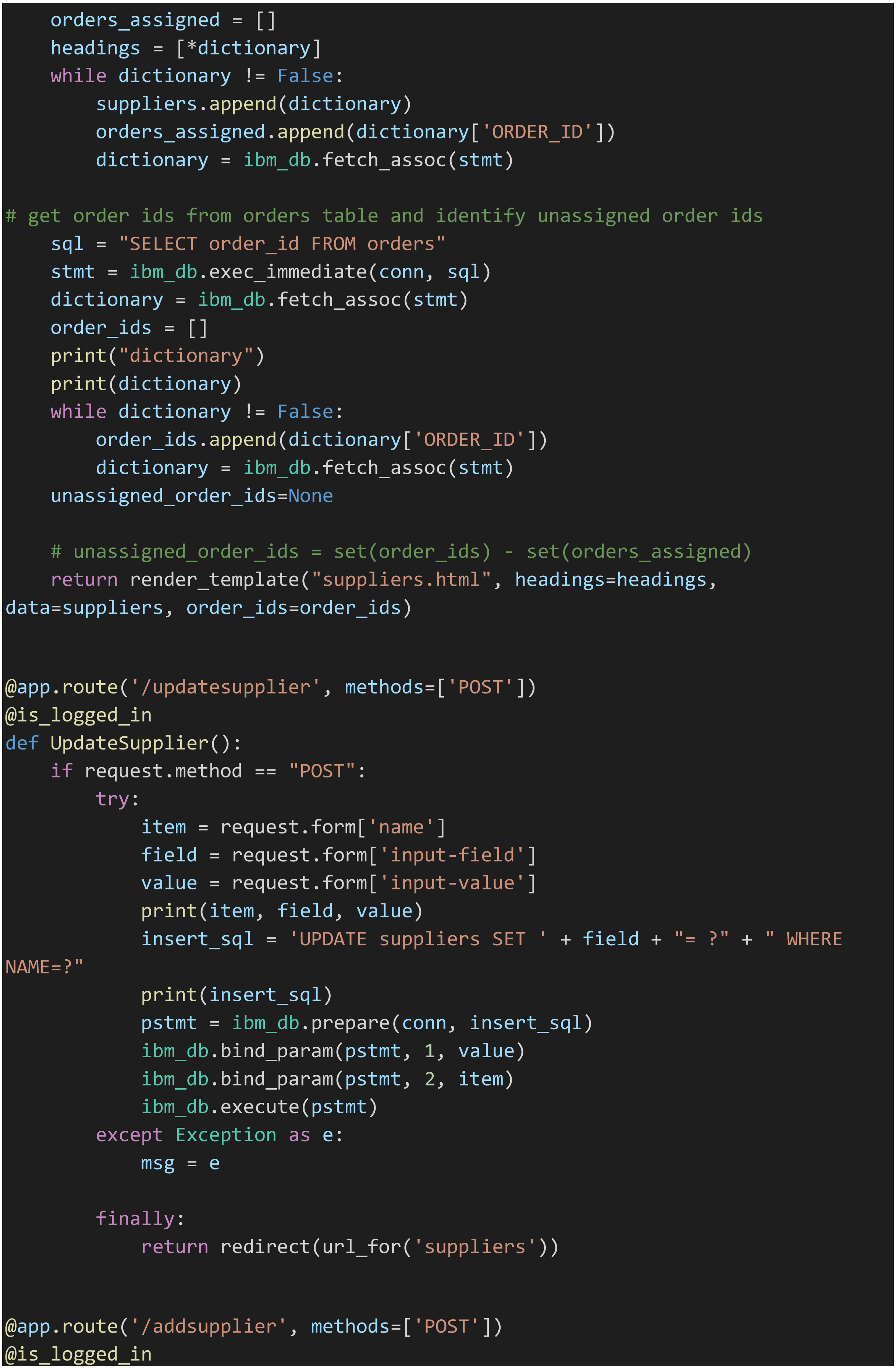
```
@is_logged_in
def updateUser():
    if request.method == "POST":
        try:
            email = session['username']
            field = request.form['input-field']
            value = request.form['input-value']
            insert_sql = 'UPDATE users SET ' + field + '= ? WHERE username=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, email)
            print(pstmt)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)
            msg = e

        finally:
            if field == 'USERNAME':
                session['username'] = value
            return redirect(url_for('profile'))


@app.route('/update-password', methods=['POST', 'GET'])
@is_logged_in
def updatePassword():
    if request.method == "POST":
        try:
            email = session['username']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM  users WHERE username=? AND
PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE username=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
        finally:
```



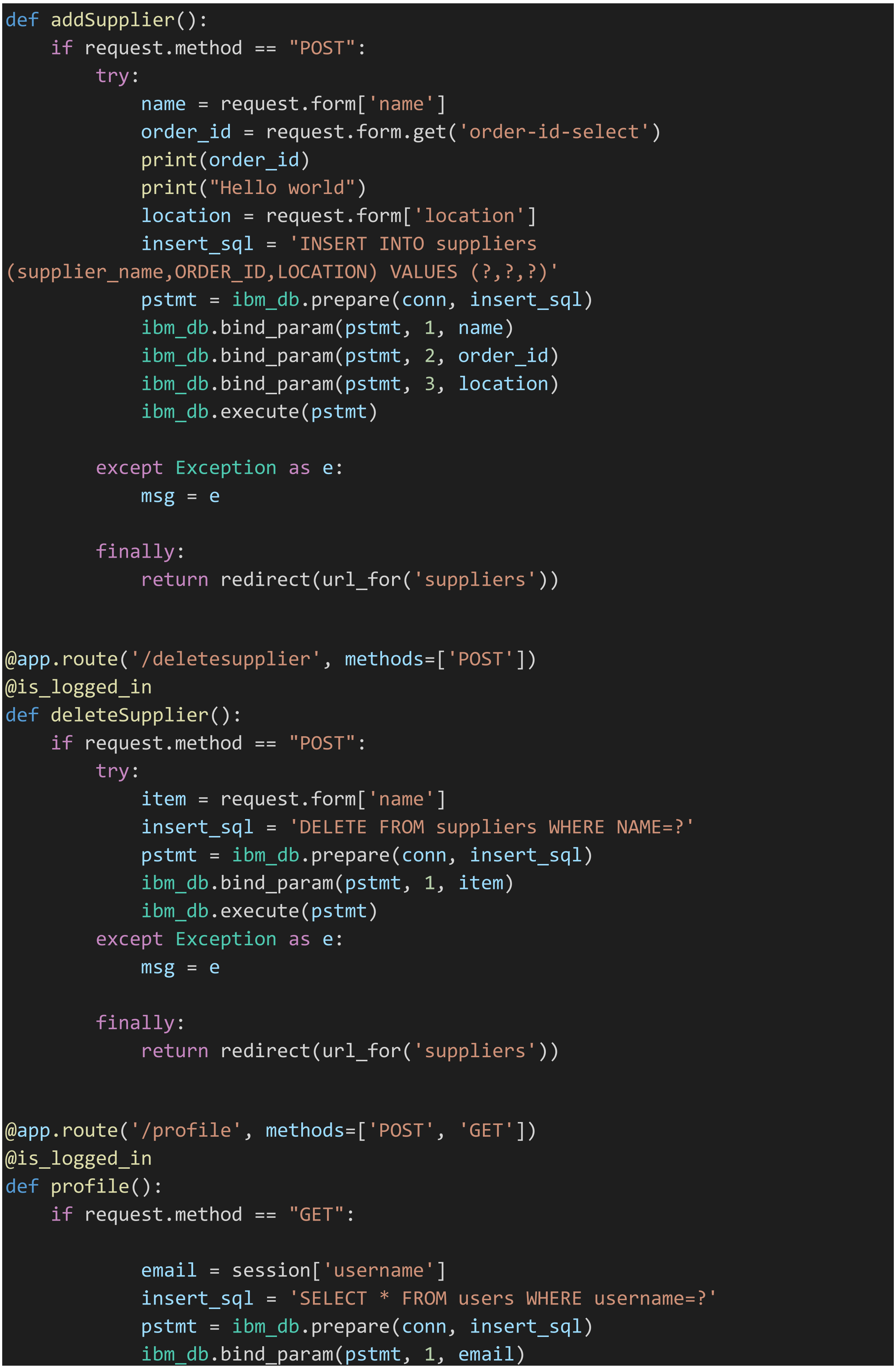
```
            return render_template('result.html')


@app.route('/orders', methods=['POST', 'GET'])
@is_logged_in
def orders():
    query = "SELECT * FROM orders"
    stmt = ibm_db.exec_immediate(conn, query)
    dictionary = ibm_db.fetch_assoc(stmt)
    orders = []
    headings = [*dictionary]
    while dictionary != False:
        orders.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template("orders.html", headings=headings, data=orders)


@app.route('/createOrder', methods=['POST'])
@is_logged_in
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']
            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['PRICE_PER_QUANTITY'])
                query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?,?,?,?,?)'
                pstmt = ibm_db.prepare(conn, query)
                ibm_db.bind_param(pstmt, 1, stock_id)
                ibm_db.bind_param(pstmt, 2, quantity)
                ibm_db.bind_param(pstmt, 3, date)
                ibm_db.bind_param(pstmt, 4, delivery_date)
                ibm_db.bind_param(pstmt, 5, price)
                ibm_db.execute(pstmt)
        except Exception as e:
            print(e)
```



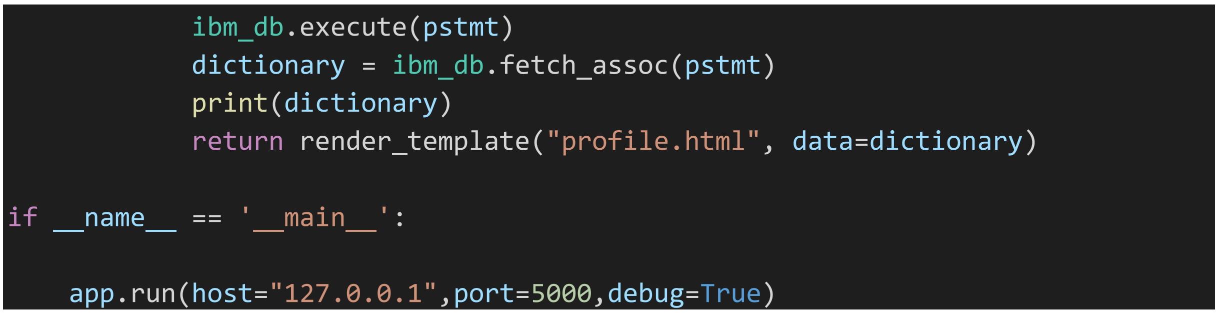
```python
        finally:
            return redirect(url_for('orders'))


@app.route('/updateOrder', methods=['POST'])
@is_logged_in
def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + "= ?" + " WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))


@app.route('/cancelOrder', methods=['POST'])
@is_logged_in
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))


@app.route('/suppliers', methods=['POST', 'GET'])
@is_logged_in
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
```



```
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

# get order ids from orders table and identify unassigned order ids
    sql = "SELECT order_id FROM orders"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    order_ids = []
    print("dictionary")
    print(dictionary)
    while dictionary != False:
        order_ids.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)
    unassigned_order_ids=None

    # unassigned_order_ids = set(order_ids) - set(orders_assigned)
    return render_template("suppliers.html", headings=headings,
data=suppliers, order_ids=order_ids)


@app.route('/updatesupplier', methods=['POST'])
@is_logged_in
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE
NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/addsupplier', methods=['POST'])
@is_logged_in
```



```
def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']
            insert_sql = 'INSERT INTO suppliers
(supplier_name,ORDER_ID,LOCATION) VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/deletesupplier', methods=['POST'])
@is_logged_in
def deleteSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))


@app.route('/profile', methods=['POST', 'GET'])
@is_logged_in
def profile():
    if request.method == "GET":

            email = session['username']
            insert_sql = 'SELECT * FROM users WHERE username=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
```



```
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            return render_template("profile.html", data=dictionary)

if __name__ == '__main__':

    app.run(host="127.0.0.1",port=5000,debug=True)
```

13.2 GITHUB AND PROJECT DEMO LINK

https://github.com/IBM-EPBL/IBM-Project-39229-1660401238